Updating Legacy Databases through Wrappers: Data Consistency Management

Thiran, Philippe; Houben, G.-J.; Hainaut, Jean-Luc; Benslimane, D.

Published in:
Proc. of the 11th Working Conference on Reverse Engineering

Publication date:
2004

Document Version
Peer reviewed version

Updating Legacy Databases through Wrappers: Data Consistency Management

Philippe Thiran, Geert-Jan Houben
Technische Universiteit Eindhoven
The Netherlands
{ph.thiran,g.j.houben}@tue.nl

Jean-Luc Hainaut
Université de Namur
Belgium
jlh@info.fundp.ac.be

Djamal Benslimane
Université de Lyon 1
France
d.benslimane@liris.cnrs.fr

Abstract

Wrapping databases allows them to be reused in earlier unforeseen contexts, such as Web-based applications or federated systems. Data wrappers, and more specifically updating wrappers (that not only access legacy data but also update them), can provide external clients of an existing (legacy) database with a neutral interface and augmented capabilities. For instance, a collection of COBOL files can be wrapped in order to allow external application programs to access them through a relational, object-oriented or XML interface, while providing referential integrity control. In this paper, we explore the principles of a wrapper architecture that addresses the problems of legacy data consistency management. The transformational paradigm is used as a rigorous formalism to define schema mappings as well as to generate as much as possible of the code of the wrappers. The generation is supported by an operational CASE tool.

1. Introduction

Legacy data systems often contain essential information that is buried in legacy databases/files structures and in application code. In many cases, these systems are the only repository of years of business rules, historical data, and other valuable information. Accessing this information is of vital importance to new open environments like the Web and to system integration in general.

A wrapper often is used to extend the life of components of existing data systems by facilitating their integration into modern distributed systems. In general, data wrapping is a very attractive strategy for several reasons. First, it does not alter the host system of the database. Secondly, it addresses the challenge of database heterogeneity by providing a standard and common interface. This interface is made up of: (1) a *wrapper schema* of the wrapped database, expressed in some abstract *canonical*[1] data model and (2) a common query language for the data as viewed through the wrapper schema. Queries/updates on the wrapper schema are also known as *wrapper queries/updates*, while native ones directly performed on the legacy schema will be called *database queries/updates*.

In this paper, we focus on database wrappers that both query and update the actual data. Therefore, the issue of integrity control is of major importance.

1.1. Updating legacy data through wrappers

Data models in general and legacy models in particular, cannot express all the structures and properties of the real world. Limitations of the modelling concepts and information hiding programming practices lead to the incompleteness of the database schema that only contains the constructs (structures and constraints) *explicitly* expressed in the DDL code, while the other constructs are *implicitly* implemented in the program codes. For examples, while foreign keys can be explicitly declared in modern relational schemas, they are implicit (hidden) in COBOL file structures and in old relational schemas. This leads to distinguishing the *physical schema*, that includes explicit constructs only, and the *logical schema*, which, in addition, includes the implicit constructs: for instance, a logical schema of COBOL files generally includes foreign keys, while the latter cannot appear in a COBOL file physical schema. Since these foreign keys must be recovered through in-depth system analysis, wrapper development and *database reverse engineering* (DBRE), one of the goals of which is to elicit hidden structures and constraints, are closely linked. More precisely, database reverse engineering will be the first step of the wrapper development methodology described in this paper.

The legacy DBMS only manages the explicit constraints while the implicit constraints are implemented by *validation code* section scattered throughout the local application programs. In order to preserve the legacy data consis-

1 The term canonical is borrowed from the domain of Federated databases [2], to which this technology is strongly related.

tency in new distributed contexts, both types of constraints must be considered. Therefore, it is the responsibility of the wrapper to guarantee legacy data consistency by rejecting updates that violate implicit constraints. In this way, both legacy applications and new applications that update the data through the wrapper can coexist without threatening data integrity (Figure 1).

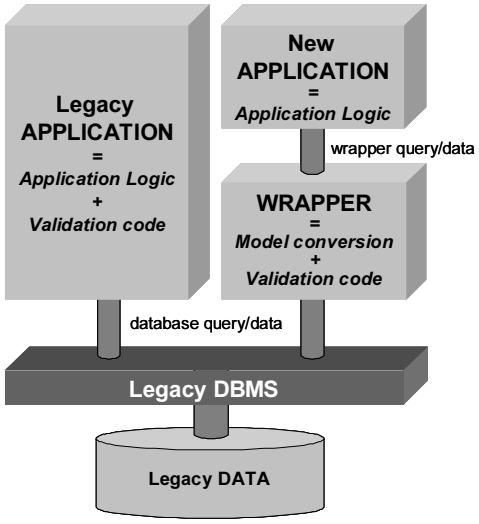


Figure 1. Coexistence of legacy and new applications. The validation code implements the implicit constraints.

1.2. Current approaches

Several prototype wrapper systems for legacy databases have been developed. They share two common characteristics against which we can compare our approach, namely the level of transparency and the update facilities they provide:

- *Wrapper transparency.* Several authors ([9], [10] and more recently, [1], [3], [4] or [11]) consider a wrapper as a pure model converter, i.e., a software component that translates data and queries from the legacy DBMS model, to another, more abstract and DBMS-independent, model. That is, the wrapper is only used to overcome the data model and query heterogeneity in database systems, leaving aside the added value that can be expected from wrapping. In such approaches, the semantic contents of both database and wrapper schemas are identical: the wrapper schema just translates explicit constructs and ignores implicit ones.
- *Update facilities.* From the application point of view, it is natural that an update request expressed on the wrapper schema be automatically mapped onto the underlying legacy database. However, very few of the current wrappers supply this kind of support (e.g., [3] or [4]). Additionally, wrappers that support updates generally do not consider implicit constraints: they only provide update query mappings without any data consistency verification.

1.3. Proposals

In this paper, we extend the work in [14] on wrappers for *legacy databases* by exploring the *updating function* of such wrappers. We consider wrappers that export a wrapper schema augmented with integrity constraints and structures that are not defined in the database schema. Updating data through such a wrapper poses the problem of guaranteeing legacy data consistency by rejecting updates that violate constraints be they implicit or explicit.

This paper describes a general architecture that addresses the problem of providing users and programmers with a wrapper able to *emulate* implicit structures and constraints. Referring to our previous work, it also states the main steps of a methodology for developing these wrappers and describes briefly a CASE tool that supports it. Both are based on the *transformational* paradigm that provides a rigorous framework for automatically translating queries and to automatically generating the data consistency management functions.

The paper is organized as follows. Section 2 develops a small case study that allows us to identify some of the main problems to be solved. Section 3 presents the main aspects of the architecture of wrappers that guarantee legacy data consistency with respect to both explicit and implicit constraints. Section 4 presents our schema transformation framework for specifying query and update mappings as well as implicit constraints. Section 5 deals with developing wrappers for legacy databases in a semi-automated way. The generation is supported by an operational CASE-tool, namely DB-MAIN. Finally, Section 6 concludes this paper.

2. Problem statement

In this section, we develop a small example that illustrates some of the problems we intend to address in this paper. We consider the sales administration of a company, which is based on a legacy relational DBMS, such as Oracle V5, in which no primary, nor foreign keys could be declared.

The economical trends force the company to make its data accessible and updatable in a new environment. The company decides to keep its database unchanged but to

build a wrapper that will allow new (and external) clients to retrieve and update the sales data. Legacy local applications are preserved while new ones can be developed through a safe interface. Indeed, new applications can no longer be forced to compensate for the weaknesses of the legacy DBMS and ensure standard integrity through its own application code as was common two decades ago. It is therefore the responsibility of the wrapper to guarantee the consistency and quality of the data flowing between the database and the new applications.

This objective raises the critical problem of developing a wrapper that allows updates that satisfy a set of constraints that can be either explicit or implicit. In particular, the wrapper should be able to make explicit, and manage, hidden constraints such as foreign and primary keys absent from the legacy model. This allows preserving the behavior of the local applications that access to the legacy database.

The main problems in building such a wrapper are to define the wrapper schema from the database schema and to define the mappings between them. For practical reason, we express the wrapper schema in a canonical extended ER model that is compliant with legacy models such as standard files, SQL-2, hierarchical and network models. However, the nature of this model is not relevant to the architecture we are discussing.

2.1. Wrapper schema definition

By analyzing the SQL DDL code, we can extract the database physical schema of the legacy database (Figure 2). The physical schema comprises the tables `Customer` and `Order`. Optional (nullable) columns are represented by the [0-1] cardinality constraint and indexes by the `acc`(ess key) constructs.

This process is fairly straightforward since it is based on the analysis of declarative code fragments or data dictionary contents. However, it recovers explicit constraints only, ignoring all the implicit constraints that may be buried into the procedural code of the programs. Hence the need for a refinement process that cleans the physical schema and enriches it with implicit constraints, providing the logical schema. Their elicitation is carried out through reverse engineering techniques such as program analysis and data analysis [5]:

- *Program analysis:* before inserting a new order, the local applications check whether the customer number is already recorded in the database (implicit foreign key).
- *Data analysis:* if `Reference` is a primary key of `Customer`, then its values must be unique, a property that will be checked through a query such as: `select * from Customer group by Reference having count(Reference)>1`.

We therefore obtain the logical schema shown in Figure 2. New constraints now appear, such as primary keys (`id` constructs), foreign keys (`ref` constructs) and redundancies (expressed by a functional dependency: `fd: Customer` $\rightarrow$ `Account`).

The next phase consists in interpreting and exporting the logical schema, therefore providing the wrapper schema through which the new client applications will view the legacy data. The logical schema expressed in an abstract data model must be expressed in the operational data model of the wrapper. In addition, this schema still includes undesirable constructs, such as redundancies and other idiosyncrasies, that must be managed but also hidden from the client applications and therefore discarded from the wrapper schema.

The logical schema of Figure 2 includes a property stating that `Address` is often split into three pertinent fragments. Moreover, it depicts a structural redundancy: the attribute `Account` of `Order` is a copy of the attribute `Account` of `Customer`. To hide this redundancy, the attribute `Account` of `Order` does not appear anymore in the wrapper schema.

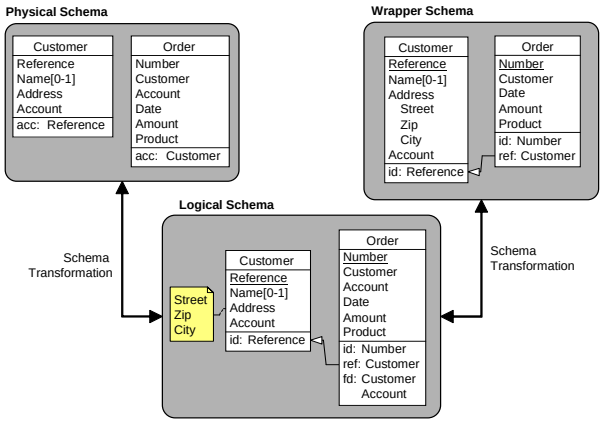


Figure 2. The physical database, logical and wrapper schemas.

2.2. Mapping definition

Once the wrapper schema has been built, we have to state how wrapper retrieval and update queries can be mapped onto legacy data. In the schema hierarchy mentioned above, the transitions between physical and wrapper schemas can be expressed as formal transformations on structures (such as discarding, renaming or aggregating) and on constraints (such as adding primary keys, foreign keys and functional

dependency). The complexity of the transformation depends on the distance between the logical and wrapper schemas. For instance, an XML-based wrapper schema would require more sophisticated mapping rules than those mentioned above [13].

The database/wrapper mappings can then be built by interpreting the transformations on structures as two-way data conversion functions whereas the implicit constraint management can be emulated by transformations on constraints. For instance, let us assume the wrapper update shown in Figure 3. By analyzing the update statement, the wrapper dynamically generates a sequence of operations that emulate and manage the implicit structures and constraints. Clearly, the main operations carried out by the wrapper are the following:

- *Implicit constraint management:* the wrapper checks the satisfaction of the constraints implied by implicit identifier 1 and the implicit foreign key 2.
- *Data error management:* the wrapper reports possible implicit constraint violation 3.
- *Redundancy management:* the wrapper controls the redundant attributes by assigning the value it gets from the source data 4.
- *Query translation:* translation of the wrapper update against the wrapper schema into updates on the physical database schema 5.

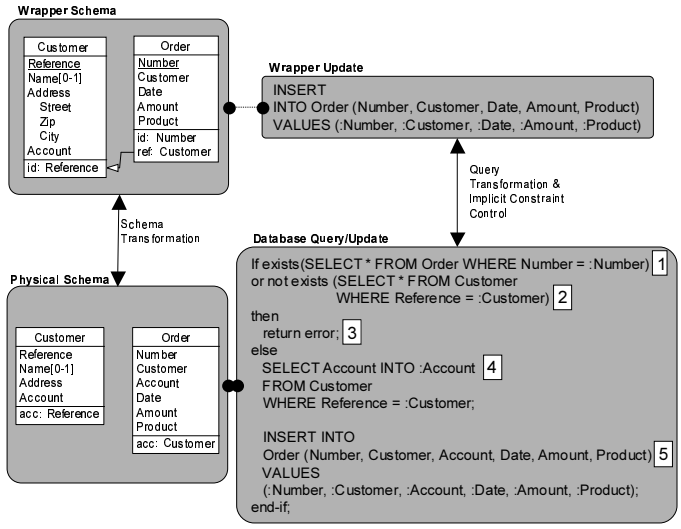


Figure 3. Example of update translation and implicit constraint management.

3. Architecture

In this section, we develop the generic architecture for wrappers that provide both extract and update facilities and that control the implicit constructs of the source databases. This leads these wrappers to emulate advanced services such as integrity control and transaction and failure management if the underlying DBMS does not support them.

The functionalities of such a wrapper are classified into functional services (Figure 4), among which we mention those which are relevant to the update aspects:

- *Wrapper query/update analysis.* The wrapper query is first analyzed so that incorrect updates are detected and rejected as early as possible.
- *Error reporting.* Wrappers report errors back to the client application.
- *Query/update and data translation.* This refers to operations that convert data and queries/updates from one model to the other.
- *Semantic integrity control.* Wrappers emulate implicit integrity constraints.
- *Security.* Data security is another important function of a wrapper that protects data against unauthorized access through its interface. Data security includes two aspects: data protection and authorization control.
- *Concurrency and recovery control.* This function is to permit concurrent updates of the underlying legacy databases. This includes transaction and failure management.

The latter two services[2] will be ignored in this paper, in which we only explain and discuss the first four classes of functionalities.

3.1. Wrapper query/update analysis

Query analysis enables rejection of queries for which further processing is either impossible or unnecessary. The main reasons for rejection are that the query is syntactically or semantically incorrect. When one of these cases is detected, the query is simply returned to the user with an explanation (see Section 3.2). Otherwise, query processing goes on. A query is incorrect if any of its attribute or entity type names are undefined in the wrapper schema, or if operations are being applied to attributes of the wrong type.

2 Concurrency, reliability and security are important additional services that are beyond the scope of this paper (see [8] for concurrency management, and [12] for security management).

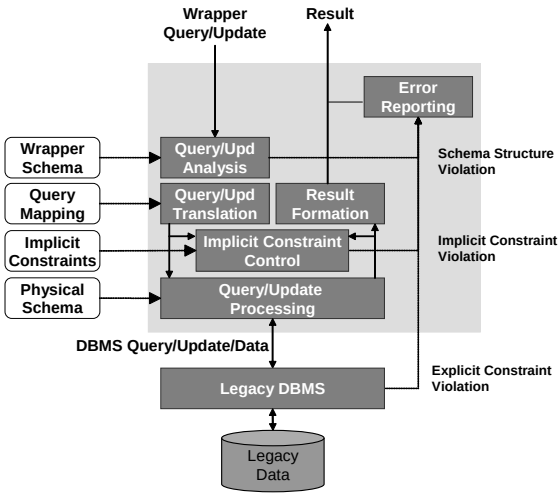


Figure 4. Wrapper architecture.

3.2. Error reporting

A wrapper returns a value that indicates the success or the failure of a wrapper query. An error can occur at two levels:

- *At the legacy DBMS level:* legacy DBMS return some indicators on completion of query execution.
- *At the wrapper level:* the wrapper catches internal errors. For example, it detects query syntax errors (see Section 3.1) or violation of implicit constraints (see Section 3.4).

Although DBMS return similar kinds of errors, each does it in a different way, so that the wrapper must provide a unified error handling interface for both DBMS and proper wrapper conditions.

3.3. Query/update and data translation

Query translation is the core function of a wrapper. It refers to operations that translate queries between two schemas (the database and wrapper schemas) and two languages (the database and wrapper query languages).

This service has been already discussed in [15], and will not be described further in this paper. It relies on the use of schema transformations for formally defining schema correspondence between the database and wrapper schemas. By replacing the schema constructs names in the wrapper query with their database equivalent, we produce a database query that can be executed on the actual data.

3.4. Implicit constraint control

While the DBMS manages the explicit constraints (i.e., constraints defined in the physical schema), the wrapper emulates the implicit constraints by rejecting updates that violate them. To prevent inconsistencies, pre-tests services are implemented in the wrapper.

This method is based on the production, at wrapper development time, of implicit constraint check components which are used subsequently to prevent the introduction of inconsistencies in the database. An implicit constraint checking is defined by a triple `<ET,T,C>` in which:

- `ET` is an entity type of the physical schema;
- `T` is an update type (`INSERT` or `DELETE`);
- and `C` is an implicit constraint assertion ranging over the entity type `ET` in an update of type `T`.

To illustrate this concept, we reconsider the example of Section 2. The implicit primary key of `Order` is associated with the triple `<Order, INSERT, C>` where `C`, defined in SQL-like expression, is `EXISTS(SELECT * FROM Order WHERE Number =:Number)`.

Implicit constraint check assertions are obtained by applying transformation rules to the wrapper schema (Section 4.2). These rules are based on the update and implicit constraint types.

Assertion enforcement efficiency. Checking consistency assertions has a cost that depends on the physical constructs implemented in the legacy database. For instance, checking uniqueness (primary or candidate key) or inclusion (foreign key) assertions can be very costly if such a construct is not supported by an index. While legacy applications easily cope with missing indexes, for instance through sort/merge batch operations, new, generally transactional, applications cannot afford relying on such outdated procedures. Besides complex architectures in which additional data structures are implemented to support efficient implicit constraint enforcement, the order in which the assertions are evaluated often is relevant.

4. Wrapper specification

Wrapping is a process that relies on schemas expressed according to different paradigms. Our previous work [6] defined a wide spectrum entity-relationship model, the so-called *Generic Entity-Relationship Model* (GER) that can express data structure schemas whatever their underlying data model and their abstraction level. For instance, the GER is used to describe physical structures such as relational data structures, as well as canonical data structures (Figure 2).

Our previous work also defined a set of primitive transformations for adding, deleting or modifying structures and

constraints to or from a GER schema. Schema transformations on GER can be applied by a developer to define correspondences (mappings) between schemas expressed in the same or in different data models. In [15], we have used these transformations for translating data, queries and updates between semantically equivalent physical and wrapper schemas.

In this section, we present the main ideas of our previous work and extend it to show how the implicit structures and constraints can be specified by schema transformations on GER.

4.1. Schema specification

For the needs of this paper, the GER can be perceived as an enriched variant of the standard entity-relationship model that encompasses current physical, logical and conceptual models. It is defined as a set of *constructs* comprising *structures* (including entity type, attribute, value domain and relationship type) and *constraints*.

Attributes can be atomic or compound, single-valued or multivalued. The roles of a relationship type can be labeled; it has a cardinality constraint (a pair of integers stating the range of the number of relationships in which any entity can appear). An attribute has a cardinality constraint too, that states how many values can be associated with each parent instance. Additional constructs such as identifiers (another name for unique keys) made of attributes and/or roles, generalized foreign keys (references) as well as existence constraints (coexistence, exclusive, at-least-one, etc.) can be defined. Constructs such as access keys, which are abstractions of such structures as indexes and access paths, and storage spaces which are abstractions of files and any other kinds of record repositories, are components of the generic model as well. Any concrete model, be it conceptual or physical, can be defines as a specialization of the GER. See [6] for more details.

4.2. Mapping specification

In [6], we define a set of transformations valid for GER schemas. These transformations can be applied by a developer to map between schemas expressed in the same or different data modeling languages. The use of GER as the unifying data model allows constructs from different data modeling languages to be mixed in the same intermediate schema (as in the logical schema of Figure 2).

A transformation consists in deriving a target schema `S'` from a source schema `S` by replacing construct `C` (possibly empty) in `S` with a new construct `C'` (possibly empty).

More formally, considering instance `c` of `C` and instance `c'` of `C'`, a transformation Σ can be completely defined by a pair of mappings `<T,t>` such that `C' = T(C)` and `c' = t(c)`. `T` is the structural mapping, that explains how to replace construct `C` with construct `C'` while `t`, the instance mapping, states how to compute instance `c'` of `C'` from any instance `c` of `C`.

4.2.1. Inverse transformation. Each transformation $\Sigma_1 \equiv$ `<`T_1`,`t_1`>` can be given an inverse transformation $\Sigma_2 \equiv$ `<`T_2`,`t_2`>`, usually denoted Σ^{-1}, such that, for any structure `C`, $T_2(T_1(C)) = C$.

So far, Σ_2 being the inverse of Σ_1 does not imply that Σ_1 is the inverse of Σ_2. Moreover, Σ_2 is not necessarily reversible. These properties can be guaranteed only for a special variety of transformations[3], called symmetrically reversible. Σ_1 is said to be a symmetrically reversible transformation, or more simply semantics-preserving, if it is reversible and if its inverse is reversible too.

From now on, unless mentioned otherwise, we will work on the structural part of transformations, so that we will denote a transformation through its `T` part.

4.2.2. Transformation sequence. A transformation sequence is a list of `n` primitive transformations: `S1-to-S2 = (T1 T2 ... Tn)`. For instance, the application of `S1-to-S2 = (T1 T2)` on a schema `S1` consists of the application of `T2` on the schema that results from the application of `T1`, so that we obtain `S2`.

As for schema transformation, a transformation can be inverted. The inverse sequence `S2-to-S1` can be derived from the sequence `S1-to-S1` and can be defined as follows: if `S1-to-S2 = (T1 T2 ... Tn)` then `S2-to-S1 = (`T_n^{-1} `...` T_2^{-1} T_1^{-1}`)` *where* T_i^{-1} is the inverse of `Ti`; and hence `S1 = S2-to-S1(S2)`. In other words, `S2-to-S1` is obtained by replacing each origin schema transformation by its inverse and by reversing the operation order.

The concepts of sequence and its inverse are used for defining the mappings between two schemas. The transformational approach then consists in defining a (reversible) transformation sequence which, applied to the source schema, produces the target schema (Section 5.1.4).

4.2.3. Transformation categories. The notion of semantics of a schema has no generally agreed upon definition. We assume that the semantics of `S1` include the semantics of `S2` iff the application domain described by `S2` is a part of the domain represented by `S1`. Though intuitive and informal, this definition is sufficient for this presentation. In this context, three transformation categories can be distinguished:

3 In [6], a proof system has been developed to evaluate the reversibility of a transformation.

- T+ collects the transformations that augment the semantics of the schema (for example adding a constraint).
- T= is the category of transformations that preserve the semantics of the schema (for example transforming a foreign key into a relationship type).
- T- is the category of transformations that reduce the semantics of the schema (for example, discarding an attribute). These transformations allow defining a wrapper schema as a subset (view) of the physical schema. As such, the usual problems associated with view updates must be addressed.

To simplify the discussion, we admit that T+ applies on constraints only whereas T= and T- can transform structures and constraints. Moreover, we ignore the transformations of type T- because they are not relevant for our discussion. As such, from now on, we will only consider transformations of types T+ and T=.

4.2.4. Implicit constraints and schema interpretation. One of the goals of T+ transformations is to make implicit constraints explicit. The other goal relevant for the problem we tackle in this paper is schema interpretation when we derive the wrapper schema.

We propose in Figure 5 and 6 two sets of representative transformational operators. The first set is made up of transformations used for schema interpretation (namely, transformation of a foreign key to a relationship type; and interpretation of multivalued attributes and compound attributes). These transformations are used in the query mapping. The second set comprises transformations dealing with implicit constraints. Due to limit space, the figure presents only two representative transformations and their associated constraint assertions.

These transformation will be used to build the wrapper schema from the physical schema, during the reverse engineering processes.

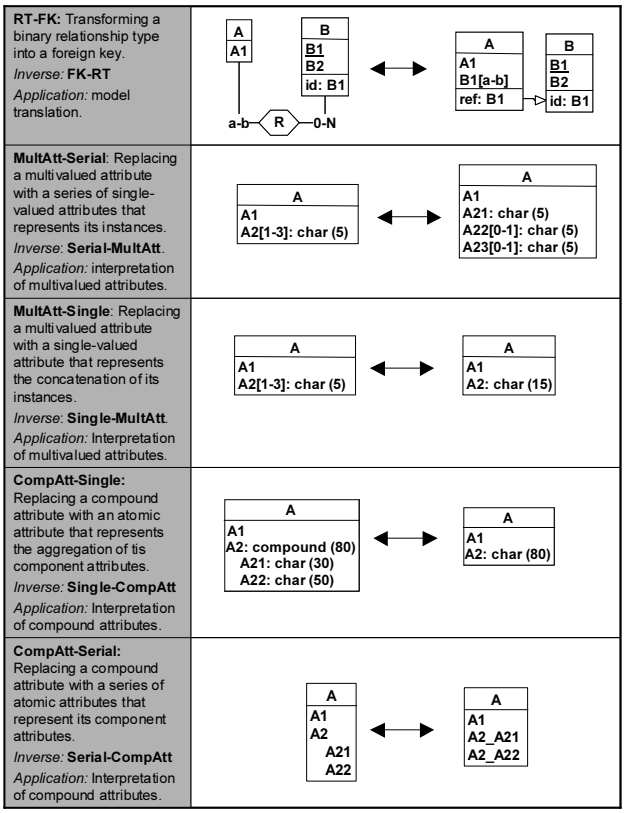


Figure 5. Five transformation operators dealing with the interpretation of three structures.

5. Wrapper development

In [14], we presented our experiences in wrapper development and our strategy for wrapper development. The key features of our approach can be summarized as follows:

- *Database reverse engineering.* Since most legacy databases have no associated documentation, the latter must first be rebuilt through database reverse engineering techniques. These techniques yield all the necessary information to specify and develop the wrappers.
- *Semi-hardcoded wrapper.* A wrapper is developed as a program component dedicated to a specific database model and to a specific database. It comprises two parts, namely a model layer, in which the aspects specific to a given data model (e.g., RDB or standard files) are coped with, and a database layer that is dedicated to the specific database schema. While the model layer is common to all the databases built in this model, the wrapper/database schemas mapping is hardcoded rather than interpreted from mapping tables as is the case in other approaches. Though such a wrapper may be larger than table-driven one for large schemas, it provides better performance.
- *Schema transformation-based wrapper generation.* In [14], the mapping rules were defined as schema transformations of type T= that are used to automatically generate the internal query mapping and the data mapping. In this paper, we use the same transformational paradigm for specifying the implicit constraint assertions that have to be controlled by the wrapper.
- *Operational CASE support.* Schema processing, mapping processing and wrapper generation is supported by a specific module of the CASE tool DB-MAIN.

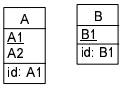
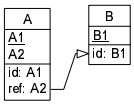
Schema Transformation	Database Schema	Wrapper Schema	Constraint Assertion	
			SQL-like Expressions	Procedural Patterns
Create-Reference: A reference constraint is added. The referencing group and the group it references are made up of existing attributes.			**<A, INSERT, C1>** *Where* C1 ≡ EXISTS(SELECT * FROM B WHERE B.B1 = :A.A2) **<B, DELETE, C2>** *Where* C2 ≡ NOT EXISTS(SELECT * FROM A WHERE A.A2 = :B.B1)	**<A, INSERT, C1>** *Where* C1 ≡ READ-FIRST B(B1=:A.A2) **<B, DELETE, C2>** *Where* C2 ≡ READ-FIRST A(A2=:B.B1)
Create-Identifier: An identifier group is added. The group is made up of existing attributes.			**<A, INSERT, C1>** *Where* C1 ≡ NOT EXISTS(SELECT * FROM A WHERE A.A1 = :A.A1)	**<A, INSERT, C1>** *Where* C1 ≡ READ-FIRST A(A1=:A.A1)

Figure 6. Transformation operators dealing with two typical implicit constraints.

5.1. Database reverse engineering for schema definition

In Section 2, we have mentioned that the database schema, merely derived from the DDL code, most often is incomplete, and must be enriched with hidden constructs and constraints made explicit. To accomplish this, we build on a proven approach, namely the DB-MAIN DBRE methodology. The key feature of this approach is threefold. Firstly, all the schemas, whatever their DBMS and their abstraction level, are expressed in the GER. Secondly, it uses the same transformational approach than that of this paper. Thirdly, this approach is supported by an operational CASE tool.

The execution of the methodology produces two result types (Figure 7): (1) the wrapper schema, including implicit and explicit constructs expressed in the canonical data model; and (2) the schema transformation sequences (and their inverse) applied on the physical schema to get the wrapper schema.

Since this methodology has been presented in former papers ([5] and [14]), we will only recall those of its processes that are relevant to wrapper development.

5.1.1. Physical extraction. This phase consists in recovering the (existing) physical schema made up of all the structures and constraints explicitly declared. This process is often easy to automate since it can be carried out by a simple parser which analyses the DDL texts, extracts the data structures and expresses them as the physical schema.

5.1.2. Logical extraction. The physical schema is a rich starting point that can be refined through the analysis of the other components of the other applications (views, subschemas, screen and report layouts, fragments of documentation, program execution, data, etc.), This schema is then refined through specific analysis techniques [5] that search non declarative sources of information for evidence of implicit constraints. The end products of this phase are (1) the logical schema `LS` that contains explicit and implicit constraints; and (2) the schema transformation sequence `PS-to-LS` (made of transformations of type `+`).

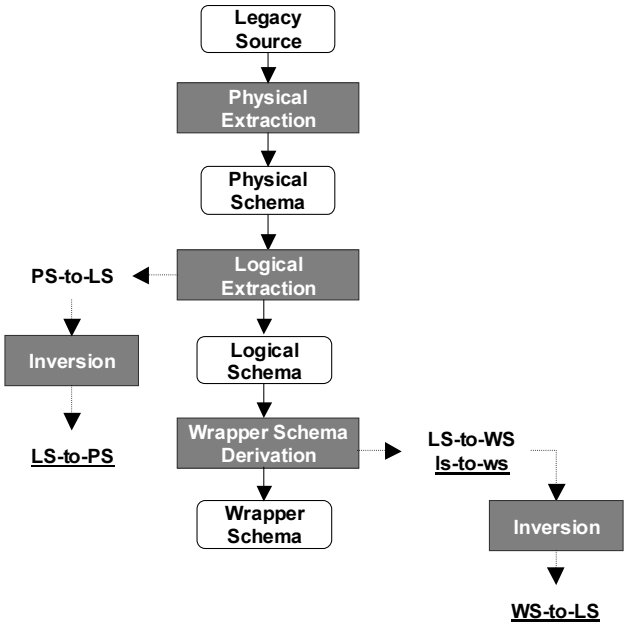


Figure 7. The DBRE process yields schemas and mappings. Underlined, the three mappings needed to develop the wrapper.

5.1.3. Wrapper schema derivation. This process of semantic interpretation consists in exporting and interpreting the logical schema, from which one tries to extract the wrapper schema `WS` and the schema transformation sequence `LS-to-WS`. Two main different problems have to be solved

through specific techniques and reasonings:

- *Model translation:* the logical schema expressed in the GER must be expressed in the operational data model of the wrapper. This process can be fairly straightforward if the logical and wrapper models are similar (e.g., DB2-to-ODBC), but it can be quite complex if they are different (e.g., Oracle-to-XML or COBOL-to-relational). This model translation basically is a schema transformation. It consists in translating a schema expressed in a source data model `Ms` into a schema expressed in a target data model `Mt` where `Ms` and `Mt` are defined as two submodels (i.e., subsets) of the generic data model. Model transformation is defined as a model-driven transformation within the GER. A model-driven transformation consists in applying the relevant transformations on the relevant constructs of the schema expressed in `Ms` in such a way that the final result complies with `Mt` (see [6] for more details).
- *De-optimization:* most developers introduce, consciously or not, optimization constructs and transformations in their physical schemas. These practices can be classified in three families, namely structural redundancies (adding derivable constructs), unnormalization (merging data units linked through many-to-one relations) and restructuring (such as splitting and merging tables). The de-optimization process consists in identifying such patterns, and discarding them, either by removing or by transforming them.

5.1.4. Mapping definition. The production of the wrapper schema (`WS`) from the physical schema (`PS`) defined in two non-necessarily distinct models, can be described by the sequence of transformations: `PS-to-WS` in such a way that: `WS = PS-to-WS(PS)` where `PS-to-WS = (PS-to-LS LS-to-WS)`.

These transformation sequences have inverse `LS-to-PS` and `WS-to-LS` such that: `WS-to-PS = WS-to-LS LS-to-PS` and `PS = WS-to-PS(WS)`.

Now we are able to specify the query/data translation and the integrity control of a wrapper (Figure 8). They rely on:

- `WS-to-LS` for the query translation (extract and update) and `ls-to-ws` for the data translation.
- `LS-to-PS` for the implicit constraint emulation.

5.2. Wrapper generation support

The wrapper generation is supported by the DB-MAIN tool. DB-MAIN is a graphical, general-purpose, programmable, CASE environment dedicated to *database application engineering*. Besides standard functions such as schema entry, examination and management, it includes advanced processors such as DDL parsers, transformation toolboxes, reverse engineering processors and schema analysis tools. An interesting feature of DB-MAIN is its complete development language, *Voyager 2*, through which new functions and processors can be developed and seamlessly integrated into the tool. Further details on DB-MAIN can be found in [7]. In the limited scope of this paper, we describe some of the DB-MAIN assistants dedicated to schema definition and wrapper code generation only.

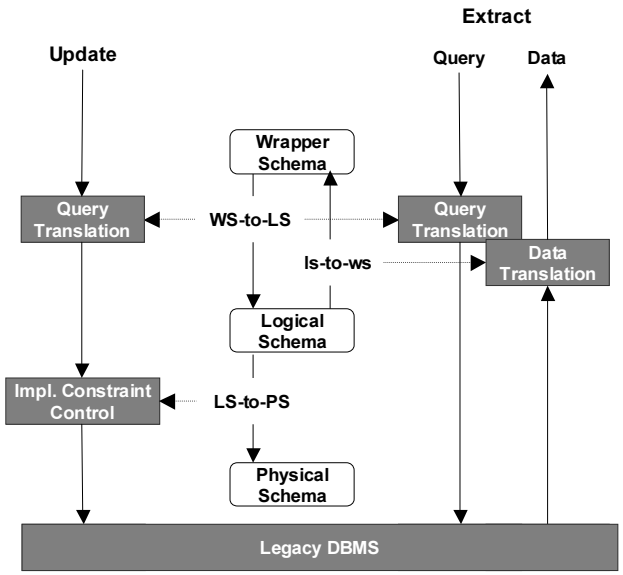


Figure 8. Three mappings built during the reverse engineering process are used to develop major components of the wrapper.

5.2.1. Schema definition

Extraction facilities. Database schemas can be extracted by a series of processors. These processors identify and parse the declaration part of the source texts, or analyze catalog tables, and create corresponding abstractions in the repository. Extractors have been developed for SQL, COBOL, CODASYL, IMS/DL1, RPG and XML DTD data structures. Additional extractors can be developed easily thanks to the *Voyager 2* environment.

Logical extraction and wrapper schema derivation. These processes heavily rely on transformation techniques. For some fine-grained reasonings, precise surgical transformations have to be carried out on individual constructs. This is a typical way of working in refinement tasks. In the case

of model translation, some heuristics can be identified and materialized into a transformation plan. DB-MAIN offers a dozen predefined model-based transformations including ER, SQL, COBOL and XML translation and untranslation.

5.2.2. Wrapper code generation

History analyzer. DB-MAIN automatically generates and maintains a history log of all the transformations that are applied when the developer carries out any engineering process such as wrapper schema definition. This history is completely formalized in such a way that it can be analyzed, transformed and inverted. A history basically is a procedural description of inter-schema mappings. The history analyzer parses history logs and transforms them into non-procedural annotations that define the inter-schema object mappings.

Wrapper encoders. The wrappers are automatically generated with two server protocols, namely SQL-based through a variant of JDBC, and object-based. At the current time, *Voyager 2* wrapper encoders for COBOL files and relational data structures are available. The generated wrappers can automatically emulate the set of schema transformations depicted in Figures 5 and 6. As such, they can emulate implicit constraints such as primary keys and foreign keys or implicit structures such as multivalued or compound attributes.

6. Conclusions

Wrapping (legacy) databases has been studied for years, leading to prototypes that insure a data model and query independence in database systems. The contribution of this paper is on the data consistency aspects of database wrapping. This paper has explored the principles of a wrapper architecture that provides update through a wrapper schema that integrates not only explicit but also implicit structures and constraints. Since the elicited constraints have been coded in the wrapper, newly developed programs can profit from automatic constraint management that the underlying DBMS cannot ensure.

One of the most challenging issues was building the inter-schema mappings. Thanks to a formal transformational approach to schema engineering, it was possible to build the mappings that derive the query decomposition and the data recomposition as well as the implicit constraint checking emulation.

The wrapper development is supported by a CASE tool that gives the developer an integrated toolset for schema definition, inter-schema mappings definition and processing, and code generation.

An experiment has been carried out in application areas of federated databases (for a city administration system) and data migration (from legacy databases to XML [13]). We are now integrating our wrapper technology into a development environment for business-to-customer applications that are built on top of legacy databases.

References

[1] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vinci. Retrieving and integrating data for multiple sources: the MOMIS approach. *Data and Knowledge Engineering*, 36, 2001.

[2] A. Bouguettaya, B. Benatallah, and A. Elmagarmid. *Interconnecting Heterogeneous Information Systems*. Kluwer Academic Press, 1998.

[3] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB (Informal Proceedings)*, pages 105–110, 2000.

[4] M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and XML. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[5] J.-L. Hainaut. Introduction to database reverse engineering. Technical report, University of Namur, 2002.

[6] J.-L. Hainaut. *Transformation of Knowledge, Information and Data: Theory and Applications*, chapter Transformation-based Database Engineering. IDEA Group, 2004.

[7] J.-M. Hick, V. Englebert, J. Henrard, D. Roland, and J.-L. Hainaut. The DB-MAIN database engineering CASE tool (version 6.5) - functions overview. DB-MAIN technical manual, Institut d'informatique, University of Namur, 2002.

[8] R. Lawrence, K. Barker, and A. Adil. Simulating MDBS transaction management protocols. In *Computer Applications in Industry and Engineering (CAINE-98) Las Vegas, Nevada*, 1998.

[9] E. Lim and H. Lee. Export database derivation in object-oriented wrappers. *Information and Software Technology, Elsevier*, 41, 1999.

[10] M. Roth and P. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. VLDB Conference, 1997.*, 1997.

[11] J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *Proceedings of the 27th VLDB Conference*, 2001.

[12] T. Souder and S. Mancoridis. A tool for securely integrating legacy systems into distributed environment. In *WCRE Proceedings*, 2000.

[13] P. Thiran, F. Estievenart, J.-L. Hainaut, and G. Houben. Exporting databases in xml - a conceptual and generic approach. In *Proc. of the CAiSE Workshops: Web Information System Modelling*, 2004.

[14] P. Thiran and J.-L. Hainaut. Wrapper development for legacy data reuse. In *WCRE Proceedings*, 2001.

[15] P. Thiran, J.-L. Hainaut, and G.-J. Houben. Wrapper for legacy databases - towards automation. In *Technical Report of the Eindhoven University of Technology*, 2004.